11/22/2011

Text Files and Streams

The class ifstream is used for **input** from a **file** and ofsteam for **output** to a **file**.

Input from a Text File

To input data from a file you first need to create a stream and associate it with a file so that the stream knows where to get the characters from. To do so, you'll need to use the **open** command and provide a filename.

(Ex) The following program reads then displays 3 numbers from a file.

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
 int i;
 ifstream infile;
 infile.open("test.txt"); // test.txt contains 3 ints
 for (int j=0;j<3;j++)
 {
    infile >> i;
    cout << i  << endl;
 }
 infile.close();
}
```

Q: what is the potential drawback for this program?

A:

(Stream State: From A to A++)

You can check the **stream state** to find whether the most recent stream operation succeeded or failed. For file streams, this includes checking the success of an attempt to open a file.

For example, attempting to open a non-existent file for input sets `failbit`. So you could check the status via:

```
ifstream fin;
fin.open("GhostFile.txt");
if (fin.fail()) // open attempt failed
{
    ...
}
```

Or, because an `ifstream` object, like an `istream` object, is converted to a `bool` type, you could use this:

```
if (!fin) // open attempt failed
```

Newer C++ implementations have a better way to check whether a file has been opened—the `is_open()` method in which an inappropriate file mode can be detected:

```
if (!fin.is_open()) // open attempt failed
```

In short, there are three ways to check whether a file has been opened and all work well. Since this operation is somehow routine, we can write a simple function `open_file` to deal with it. Our function takes references to an `ifstream` and a `string`. The `string` holds the name of a file to associate with the given `ifstream`:

```
// opens in binding it to the given file
ifstream& open_file(ifstream& in, const string& file)
{
   in.close();     // close in case it was already open
   in.clear();     // clear any existing errors
   // if the open fails, the stream will be in an invalid state
   in.open(file.c_str()); // open the file we were given
   return in; // condition state is good if open succeeded
}
```

Because we do not know the stream state, we start by calling **close** and **clear** to put the stream into a valid state. We next attempt to open the given file. If the open fails, the stream's condition state will indicate that the stream is unusable. We finish by returning the stream, which is either bound to the given file and ready to use or is in an error condition.

Input Stream Processing

Once we successfully open an input stream, we can use input operator `>>` (pretty much as cin) to process input file stream. You will in general use `while` loop to read the contents of input file stream (e.g., `fin`):

```
while (fin >> n)
```

Tests are `true` only if the stream state is good (all bits cleared). The same tests can also apply for `cin`.

```
while (cin >> n)
```

(**Ex**) given the code fragments and data file

ints.dat

```
12 -34
```

```
 7
```

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>  // for system command
using namespace std;

// opens in binding it to the given file
ifstream& open_file(ifstream& in, const string& file)
{
   in.close();     // close in case it was already open
   in.clear();     // clear any existing errors
   // if the open fails, the stream will be in an invalid state
   in.open(file.c_str()); // open the file we were given
   return in; // condition state is good if open succeeded
}

int main()
{
   ifstream ints_in;
   int n;
   string file_name;
   cout << "Enter the file name: ";
   cin >> file_name;
   if (!open_file(ints_in, file_name)) {
      cerr << "Complain: I cannot find the file" << endl << endl;
      cerr << system("dir") << endl; //only for windows OS
      return -1;
   }
   cout << "Contents of ints.dat:\n";
   while (ints_in >> n)
       cout << n << ' ';
   return 0;
}
```

Footnote: C and C++ String

When we do:

```
ifstream infile;
infile.open("test.txt");
```

The string literal "test.txt" is actually a C string (its type is const char *). If you want to use C++ string, you need to convert C++ string to C string.

```
string file_id;
ifstream wage_in;
cout << "Please enter file ID: ";
cin >> file_id;
wage_in.open(file_id.c_str());
```

The argument passed to open is `file_id.c_str()` rather than simple `file_id`. The reason is because open requires an argument whose type is an “old style” C string, not a C++ string.